removed from the buffer pool. Thus, the block previously identified as block (Z), the block receiving the backward block pointer update, is now identified as block (Y).

BWD(Y)=L(X);

Mark the block modified.

MOD(2)=ON;

(8) REMOVE SHIFTED SOURCE ENTRIES

This process step occurs during the resynchronization phase of the compression cycle. For this transformation, define that entry (w) is located in block (Y), and further, that the offset of (w) in (Y) is always the offset of the first entry in (Y) which does not also appear in (X). Remove all entries in block (Y) which precede entry (w). (RANGE is the number of character positions in use in the block for entry (w) and the entries which follow it).

RANGE=OL(B(Y))−O(w)+EL;

B(Y)(LENGTH(HEADER):RANGE)=B(Y)-(O(w):RANGE);

Record the new last entry offset in block (Y).

OL(B(Y))=LENGTH(HEADER)+RANGE−EL;

Mark the (Y) block modified.

MOD(2)=ON;

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

1. In a data processing system storing a plurality of discrete entities, each identified by a single parameter within a monotonic parameter spectrum, each resident at an addressable location and each locatable by searching a system maintained addresses,

a method of maintaining said multi-level index by said system comprising the steps of:

executing a sequence of processing cycles wherein each cycle progresses through the index levels in a first and then a second direction, selectively performing at each level of the index, while performing a processing cycle in said first direction, a first subset of basic operation iterations, including,

basic operations for duplicating the presence of an original parameter, and

selectively performing at each level of the index, while performing a processing cycle in said second direction, a second subset of basic operation iterations including,

basic operations for deleting the original presence of said duplicated parameter.

2. A method as set forth in claim 1 for an index structure of pointer linked blocks, said blocks each having a storage capacity for plural said parameters and plural said pointers,

said first subset of basic operation iterations including

basic operations for removing some but not all of said pointers linking one of said blocks into said index;

said second subset of basic operation iterations including basic operations for removing the remaining said pointers linking said one of said blocks into said index.

3. A method as set forth in claim 2 including, in each processing cycle,

identifying, at each said level, a window in said index, said window comprising a predetermined number of said blocks at said level and an equal number of said blocks if such exist, at the next adjacent level in said first direction, said blocks being interlinked by said pointers;

selecting basic operations in both said first and second subsets of basic operation iterations in accordance with the contents of said window;

performing said selected basic operations in said first subset of basic operation iterations within said window, and

identifying a next window in said index comprising said blocks at said next adjacent level and linked ones of said blocks in the next adjacent level thereto in said first direction.

4. A method as set forth in claim 3 including, in any interrupted processing cycle executing in said second direction, reidentifying each said window;

reselecting said basic operations of said second subset of basic operation iterations; and

performing said reselected basic operations.

5. The method as set forth in claim 1 wherein the permitted iterations are defined by action vectors contained in an action vector table, each said action vector having a vector element corresponding to each said basic operation, said vector elements being arranged in sequence order and determining the performance, when of one significance and the non-performance, when of the opposite significance, of the corresponding one of said basic operations in relation to the contents of a window in said index in its current position, said window being movable within said index between a base level of said index and an apex level thereof in said first direction, including

selecting, from said action vector table, one of said action vectors in response to the contents of said window in its current position in said index.

6. The method as set forth in claim 5 further including generating a progress vector, element by element in such a way that each element is of one significance when a predetermined relationship obtains in said window in the current position thereof and of the opposite significance when said predetermined relationship does not so obtain,

associatively accessing a progress vector table using said generated program vector as a search argument to obtain a result field,

accessing said action vector table in response to certain of said result fields and

moving said window in response to other of said result fields.

7. The method as set forth in claim 6 further including generating a condition vector, element by element, in such a way that each element is of one significance when a predetermined relationship obtains in said window in the current position thereof and of the opposite significance when said predetermined relationship does not so obtain,

associatively accessing a condition vector table using said generated condition vector as a search argument to obtain a result field,

accessing said action vector table in response to certain of said result fields and

accessing said action vector table in response to result fields from said progress vector table in response to other of said condition vector table result fields.

8. The method as set forth in claim 7 further including, for resumption after interruption,

moving said window systematically within said index, generating a condition vector for each current position of said window, detecting that one of said generated condition vectors closest to the apex level of said index that would cause an action vec-